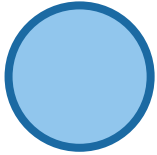


Graph Data Management

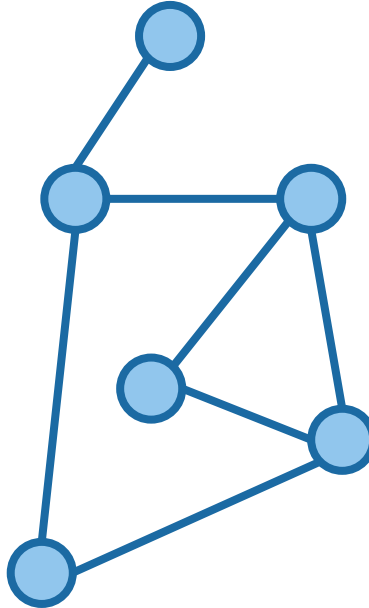
Scalable Data Engineering

Basic Graph Elements

Nodes (Dots)



- Like an entity in ER
- Exist on their own
- Have object identity



Edges (Lines)

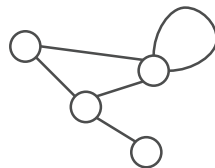


- Like a relationship in ER
- Exist only between nodes
- Identity depends on the nodes they connect

Building Blocks of Graph Data Models

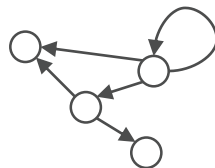
Basis

- Assuming a domain of objects $\mathcal{O}$
- A graph is a structure (V, E)
- Vertices are objects: $V \subseteq \mathcal{O}$
- Edges are 2-element subset of vertices: $E \subseteq \{\{x, y\} | x, y \in V\}$



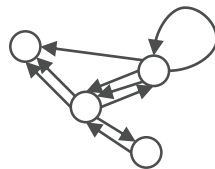
Direction

- Edge are pairs of vertices $E \subseteq V \times V$



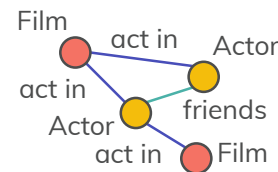
Multigraph

- A graph is a structure (V, E, ρ)
- Edges are objects: $E \subseteq \mathcal{O}$
- $\rho: E \rightarrow \{\{x, y\} | x, y \in V\}$ (undirected) / $\rho: E \rightarrow V \times V$ (directed)
is a function assigning to each edge
a 2-element subset of / ordered pair of vertices



Labels

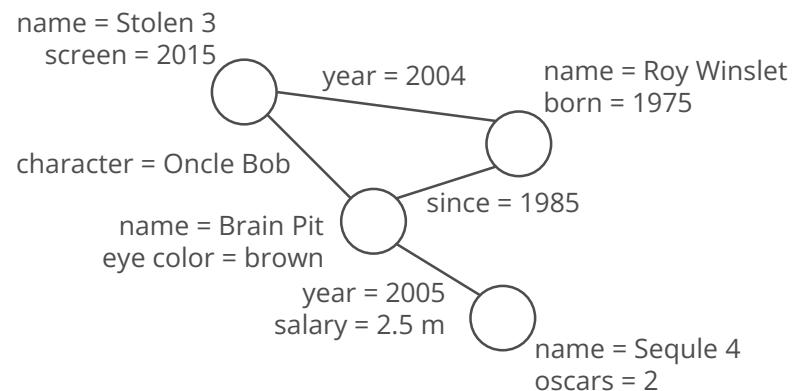
- Descriptive type information tagged to the objects
 - Indicate which kind of real world entity an object represents
 - Assuming a domain of labels $\mathcal{L}$
 - A graph is a structure (V, E, λ)
 - $\lambda: V \rightarrow \mathcal{L}$ (vertex label)
 - $\lambda: E \rightarrow \mathcal{L}$ (edge label)
 - $\lambda: V \cup E \rightarrow \mathcal{L}$ (vertex and edge label)
 - $\lambda: V \rightarrow 2^{\mathcal{L}}$ (vertex labels)
 - $\lambda: E \rightarrow 2^{\mathcal{L}}$ (edge labels)
 - $\lambda: V \cup E \rightarrow 2^{\mathcal{L}}$ (vertex and edge labels)
- is a partial / total function assigning to each vertex / edge / vertex and edge a label / a set of labels



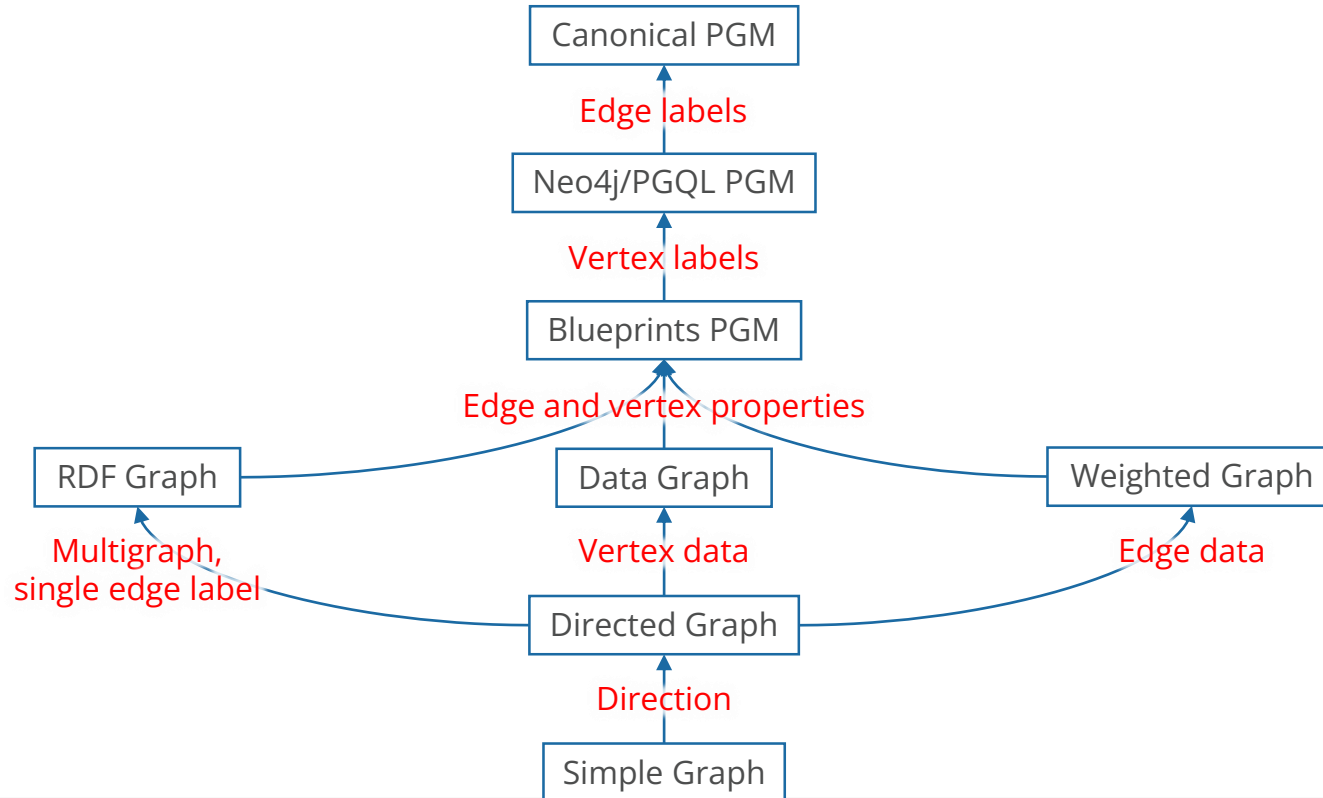
Building Blocks of Graph Data Models

Data / Properties

- Information / statements about objects
 - Rich objects, “the actual data”, “the meat”
 - Assuming a domain of data / values $\mathcal{D}$ and a domain of property keys / attributes $\mathcal{K}$
 - A graph is a structure (V, E, ν)
 - $\nu: V \rightarrow \mathcal{D}$ (vertex data)
 $\nu: E \rightarrow \mathcal{D}$ (edge data)
 $\nu: V \cup E \rightarrow \mathcal{D}$ (vertex and edge data)
 $\nu: V \times \mathcal{K} \rightarrow \mathcal{D}$ (vertex properties)
 $\nu: E \times \mathcal{K} \rightarrow \mathcal{D}$ (edge properties)
 $\nu: (V \cup E) \times \mathcal{K} \rightarrow \mathcal{D}$ (vertex and edge properties)
- is a partial function assigning data / values to each vertex / edge / vertex and edge



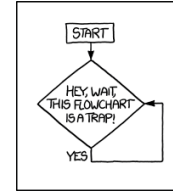
Family of Graph Data Models



Further Terminology

Loops

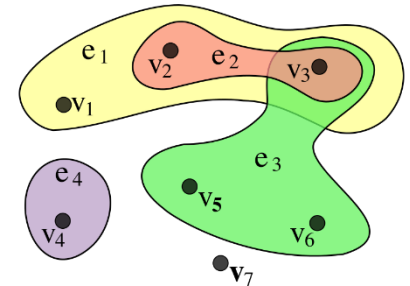
- Edges with both ends being the same vertex



[<http://xkcd.com/1195/>]

Hypergraphs

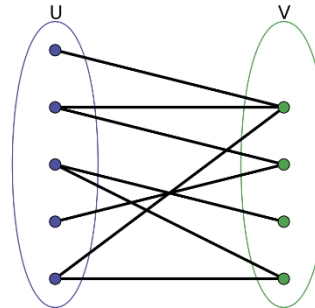
- Edges can connect more than two vertices
- $G = (V, E)$ with $E \subseteq 2^V$



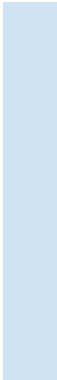
[<http://commons.wikimedia.org/wiki/File:Hypergraph-wikipedia.svg>]

Bipartite graphs

- Relation between two sets as a graph
- $G = (U, V, E)$ with $E \subseteq (U \times V) \cup (V \times U)$

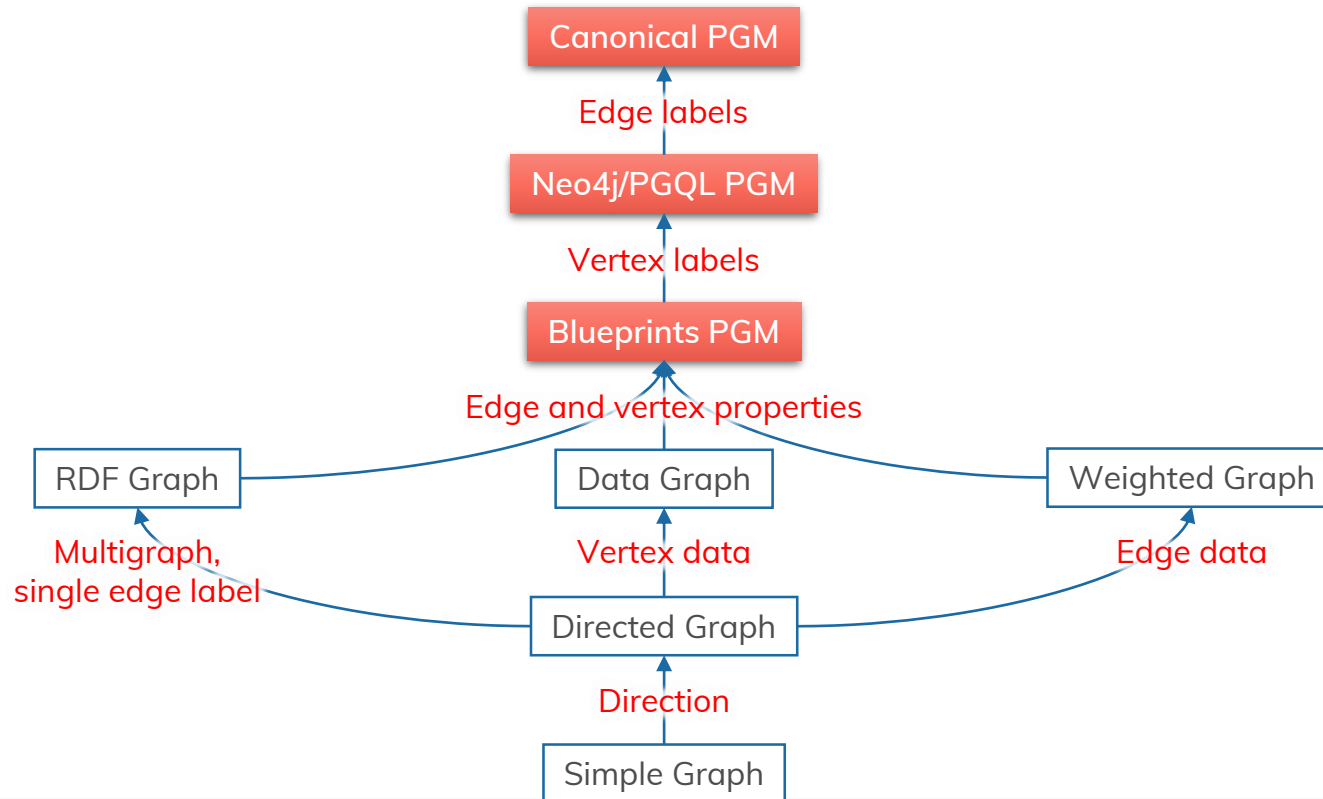


[<https://commons.wikimedia.org/wiki/File:Simple-bipartite-graph.svg>]



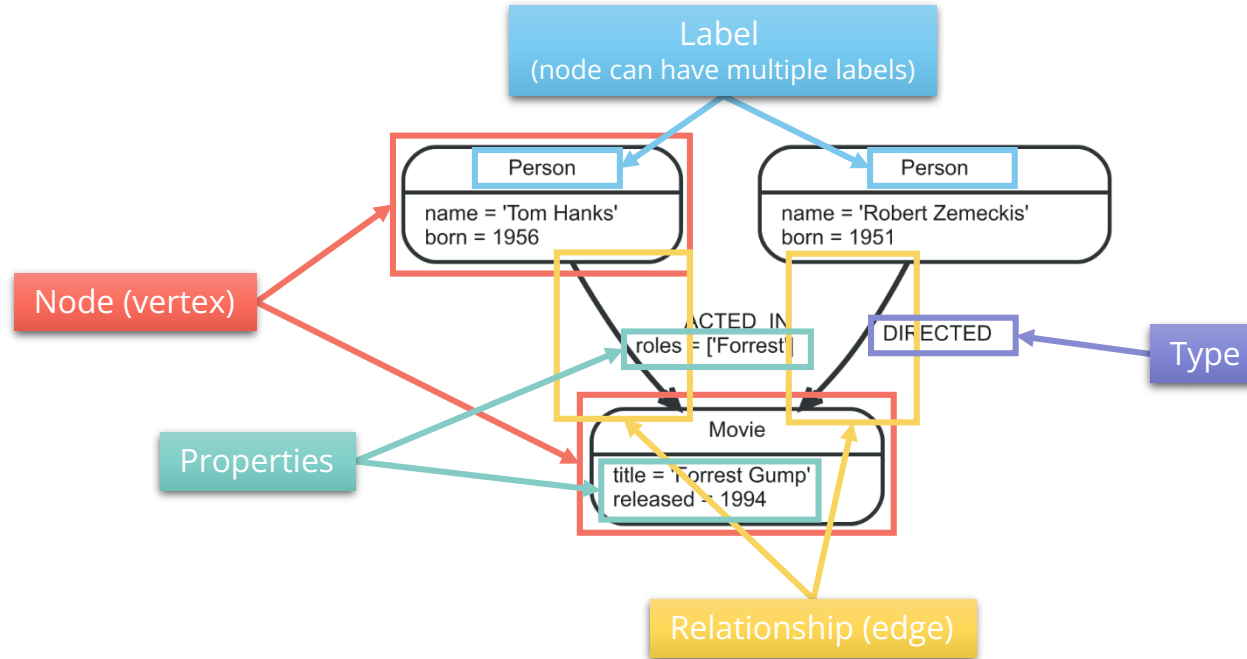
Property Graph Model (PGM)

Family of Graph Data Models



Neo4j PGM

[<http://neo4j.com/docs/developer-manual/current/introduction/#graphdb-concepts>]



Differences in Implementation

Vertex identity

- User-specified (TinkerPop) vs. system-generated (Neo4j)

Edge identity

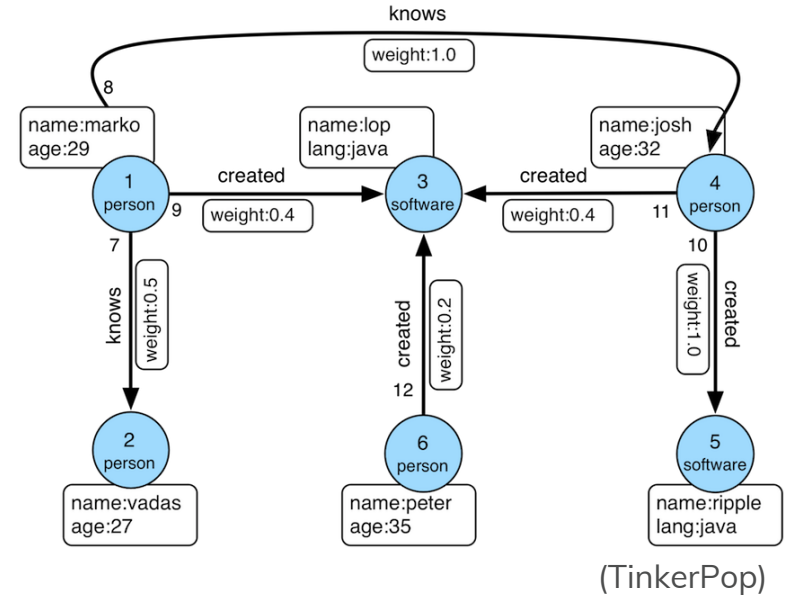
- Implicit (based on source and target node) vs. explicit
- Visible vs. not visible
- User-specified (TinkerPop) vs. system-generated (Neo4j)

Labels

- Single label (TinkerPop, Neo4j edges) vs. multiple labels (Neo4j vertices)

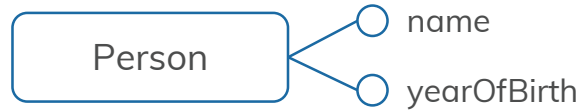
Properties

- Multi-valued properties vs. single-valued properties



Entity with Properties

- Entity relationship model

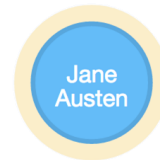
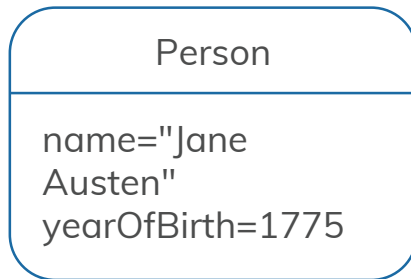


- Schema typically implicit, i.e. given with instances

- Instance

`(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })`

Cypher ASCII-art syntax

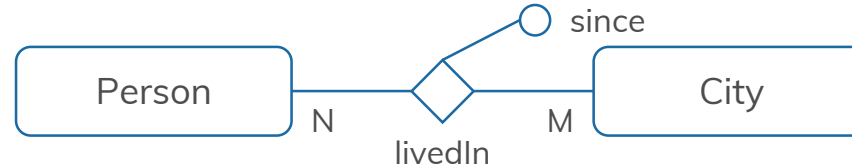


Person

<id>: 175 yearOfBirth: 1775 name: Jane Austen

Relationships (N:M)

- Entity relationship model

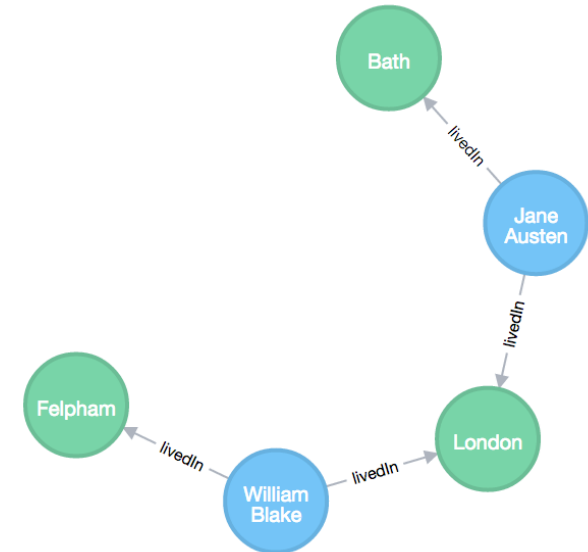


- Vertices

(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })
(wb:Person { name: 'William Blake', yearOfBirth: 1757 })
(lo:City {name: 'London'})
(ba:City {name: 'Bath'})
(fe:City {name: 'Felpham'})

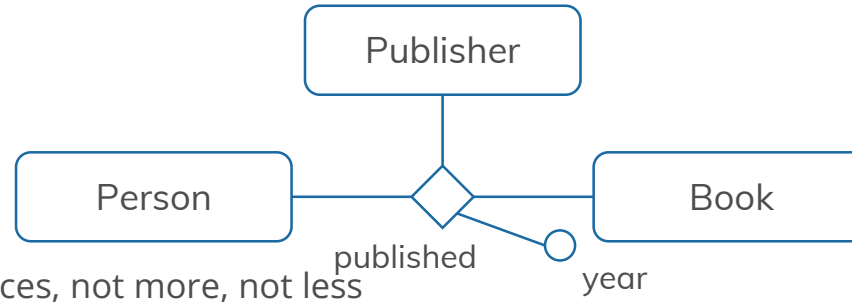
- Edges

(ja)-[:livedIn {since: 1775}]->(lo)
(ja)-[:livedIn {since: 1800}]->(ba)
(wb)-[:livedIn {since: 1757}]->(lo)
(wb)-[:livedIn {since: 1800}]->(fe)



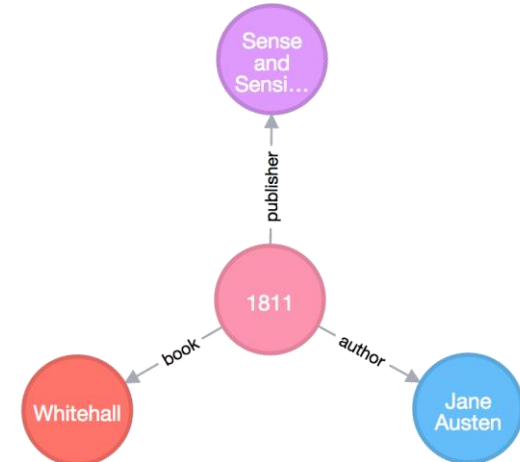
n-ary relationships (with $n > 2$)

- Entity relationship model



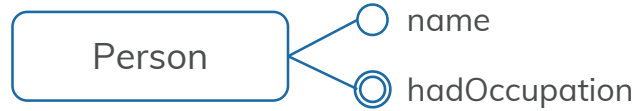
- Not directly expressible
since edge connect exactly two vertices, not more, not less
- Solution: Reification of relationship

```
(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })  
(wh:Publisher { name: 'Whitehall' })  
(sas:Book {title: 'Sense and Sensibility' })  
(pub:Publication {year: 1811 })  
(pub)-[:author]->(ja)  
(pub)-[:book]->(wh)  
(pub)-[:publisher]->(sas)
```



Multivalued properties

- Entity relationship model



- Expressibility depends on datatype system of specific system in use
- Possible in Neo4j:

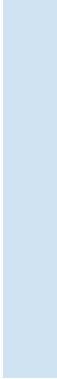
`(wb:Person {name: 'William Blake', hadOccupation: ['Poet','Painter','Printmaker']})`



Person

`<id>: 183 hadOccupation: Poet,Painter,Printmaker name: William Blake`

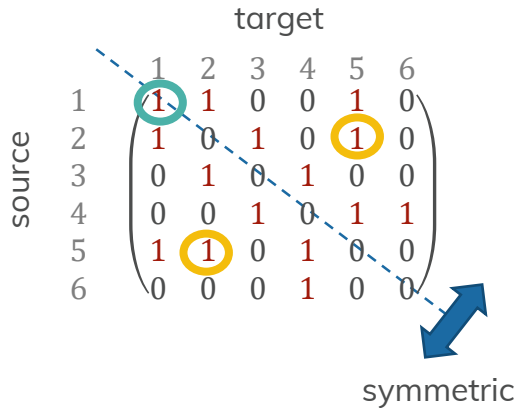
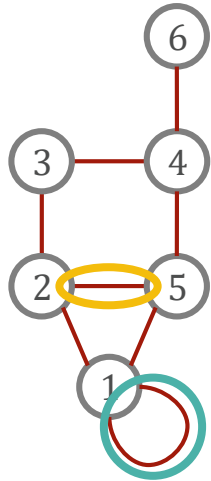
- If not expressible: Reification of values of multivalued property



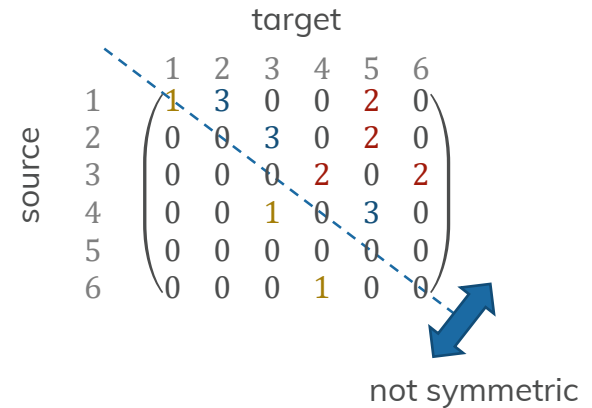
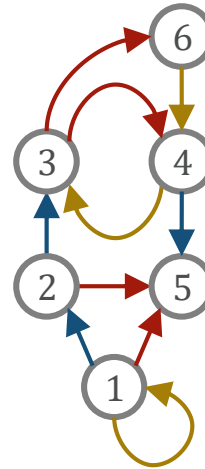
Adjacency Representation

Adjacency Matrix

Undirected graph without labels



Directed graph with edge labels



Adjacency Matrix

Formalism to define graph operations

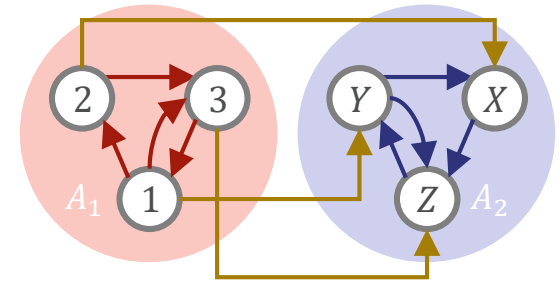
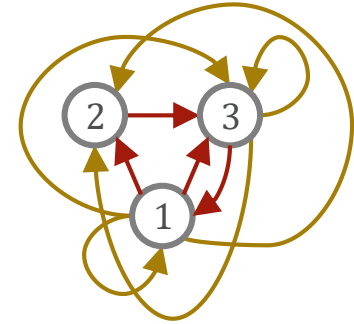
- Reachability: $A^2 = A \cdot A$ shows all paths of length 2 between nodes of graph represented by A

$$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

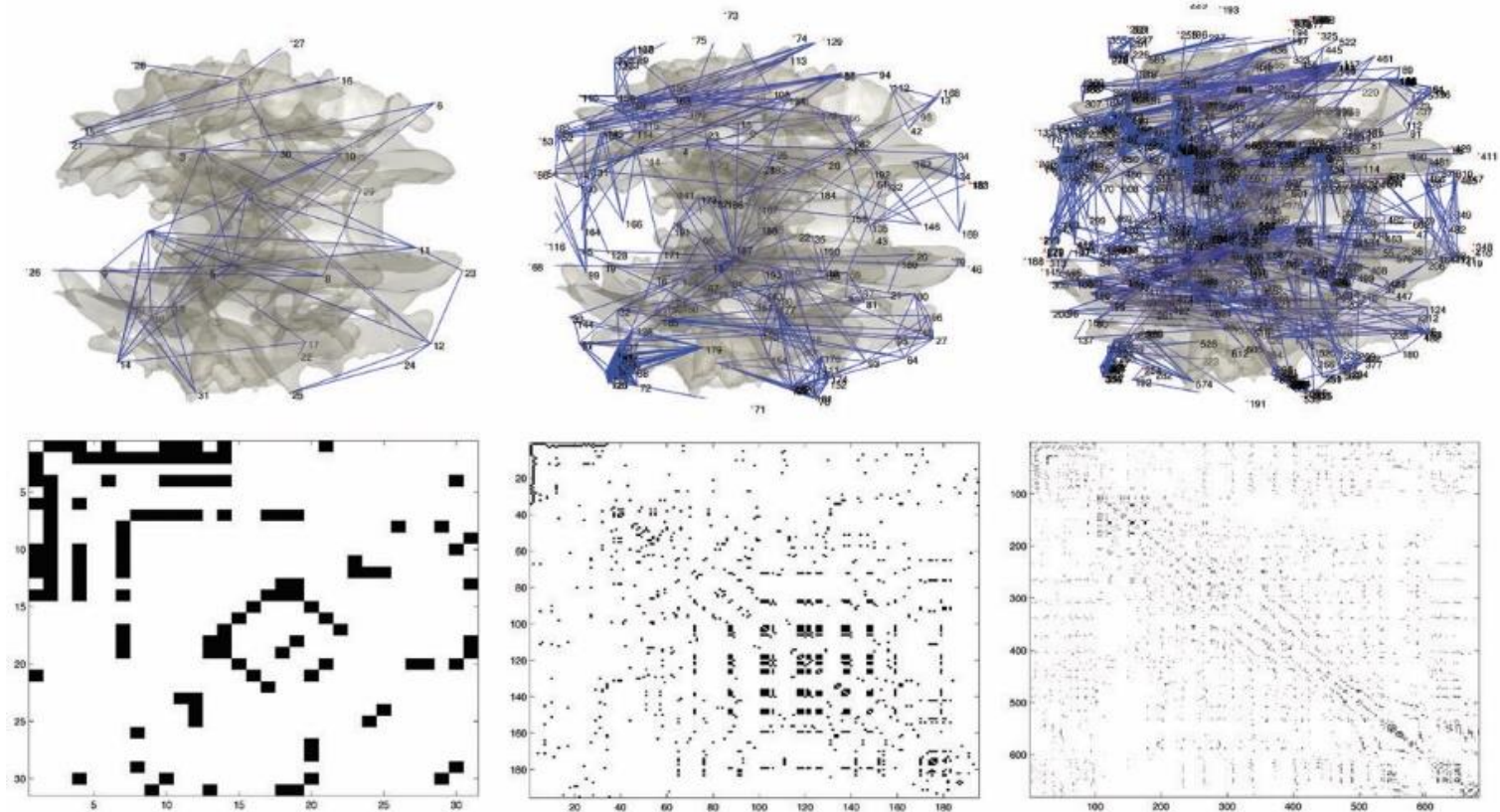
- Graph isomorphism: Graphs represented by A_1 and A_2 are isomorphic iff there exists a permutation matrix P such that $PA_1P^{-1} = A_2$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

- Note: Most definitions work only for unlabeled graphs!



Adjacency Matrix



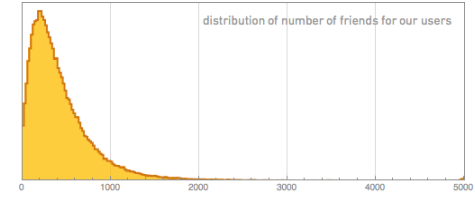
[<http://brainimaging.waisman.wisc.edu/~chung/graph/admatrix.jpg>]

Adjacency Matrix

As Storage structure

- Easy to implement with array types of most programming languages
- Represents topology -> no properties
- Not suitable for larges and sparse graphs common in data management

- Social networks, e.g. Facebook: >1.4 billion user, average user has ~350 friends
→ at least 1.400.000.000 x 1.400.000.000 matrix, with ~350 1s per matrix row



[<http://blog.stephenwolfram.com/2013/04/data-science-of-the-facebook-world/>]

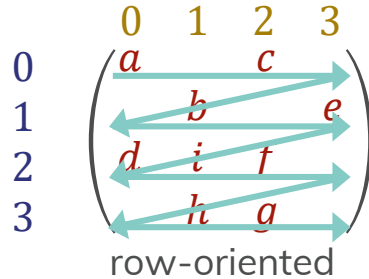
- Road networks, e.g. Road network of California: ~2 million junctions, 99% of junctions with less than six ways
→ at least 2.000.000 x 2.000.000 matrix, with less than six 1s per matrix row



- Compression necessary

Sparse Matrix Storage

Compress sparse row (CSR)



Position for row # in other two arrays:

0 1 2 3
Row position array:

0 1 2 3 4 5 6 7 8
Column index array:

Cell value array:

Sparse Matrix Storage

Compress sparse row (CSR)

$$\begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} a & & c & \\ & b & & e \\ d & i & f & \\ & h & g & \end{pmatrix} \end{matrix}$$

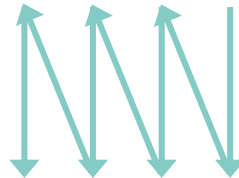
Compress sparse column (CSC)

- Like CSR
- But column oriented

Position for row # in other two arrays:

#	0	1	2	3
Row position array:	0	2	4	7

#	0	1	2	3	4	5	6	7	8
Column index array:	0	2	1	3	0	1	2	1	2
Cell value array:	a	c	b	e	d	i	f	h	g



Adjacency Lists

Compression of matrix

$$\begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} a & & c & \\ & b & & e \\ d & i & f & \\ & h & g & \end{pmatrix} \end{matrix}$$

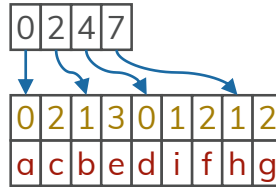
Adjacency list

- Source vertex with outgoing edges

Without edge labels

0 -> (0,2)
1 -> (1,3)
2 -> (0,1,2)
3 -> (1,2)

Compressed
Sparse Row

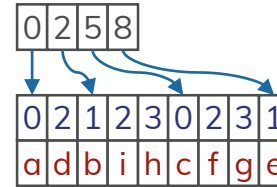


source-oriented

With edge labels

The same!!!

Compressed
Sparse Column



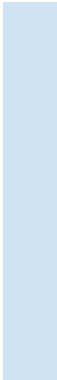
target-oriented

Almost the same!!!

With edge properties

Coordinate
list

(0,0,a)
(0,2,c)
(1,1,b)
⋮
(2,2,f)
(3,1,h)
(3,2,g)



Relational Representation

Triple Table

[Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007]

A table with three columns: subject, predicate, object

Subject	Predicate	Object
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#fullName>	"Eric Miller"
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#mailbox>	<mailto:e.miller123(at)example>
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#personalTitle>	"Dr."
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2000/10/swap/pim/contact#Person>

- Generic, simple, straightforward approach
- Can be built on relational storage engines
- Add indexing for faster lookups
- Queries need to operate on all triples
(no data portioning based on schema information)
- High storage redundancy

Vertex and Edge Table

Two universal tables

- one for the vertices, one for the edges

ID	Type	Color	Name	RAM	Nationality	Source	Target	Type	Rating
1	Product	black	"Apple iPad MC707LL/A"	64 GB		1	7	in	
...	...	...	...	...	...	...	...	...	...
4	Category		"Cell Phones & Accessories"			5	4	part of	
5	Category		"Phones"			7	6	part of	
...	...	...	...	...	...	...	...	...	...

- Requires efficient handling of NULL values
 - Given for instance in column stores
- Queries need to operate on all vertices/edges
(no data portioning based on schema information)

Property Table Approaches

[Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007]

(Clustered) property table

Property Table

Subj.	Type	Title	copyright
ID1	BookType	“XYZ”	“2001”
ID2	CDType	“ABC”	“1985”
ID3	BookType	“MNP”	NULL
ID4	DVDType	“DEF”	NULL
ID5	CDType	“GHI”	“1995”
ID6	BookType	NULL	“2004”

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	“Fox, Joe”
ID2	artist	“Orr, Tim”
ID2	language	“French”
ID3	language	“English”

- Groups properties that tend to be defined together in a table
- Example:
 - Properties: type, title, and copyright date
 - Table stores all triples with one of these properties
- Multiple property tables with different clusters of properties may be created
- One particular property may only appear in at most one property table
- Triples that not fit any property table are stored in a left-over triple table
- Multivalued attributes are problematic
- Queries that have unspecified property are problematic
- Introduce NULL values

Property Table Approaches

[Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007]

Property-class table

Class: BookType

Subj.	Title	Author	copyright
ID1	“XYZ”	“Fox, Joe”	“2001”
ID3	“MNP”	NULL	NULL
ID6	NULL	NULL	“2004”

Class: CDType

Subj.	Title	Artist	copyright
ID2	“ABC”	“Orr, Tim”	“1985”
ID5	“GHI”	NULL	“1995”

Left-Over Triples

Subj.	Prop.	Obj.
ID2	language	“French”
ID3	language	“English”
ID4	type	DVDType
ID4	title	“DEF”

- Cluster similar sets of subjects together in the same table
- Exploits the type property of subjects
- A property may exist in multiple property-class tables
- A subject may also exist in multiple tables
- Multivalued attributes are problematic
- Queries that do not select on class type are problematic
- Introduce NULL values

Property Table Approaches

[Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007]

Triple table

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

(Clustered) property table

Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

Property-class table

Class: BookType

Subj.	Title	Author	copyright
ID1	"XYZ"	"Fox, Joe"	"2001"
ID3	"MNP"	NULL	NULL
ID6	NULL	NULL	"2004"

Class: CDType

Subj.	Title	Artist	copyright
ID2	"ABC"	"Orr, Tim"	"1985"
ID5	"GHI"	NULL	"1995"

Left-Over Triples

Subj.	Prop.	Obj.
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

Reduce numbers subject-subject self joins necessary to reconstruct entities

Property Table Approaches

[Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007]

Example query

- “find all items in the catalog copyrighted before 1990 in a language other than English”

```
SELECT ?i ?l  
WHERE {  
  ?i <copyright> ?c .  
  FILTER ?c < 1990 .  
  ?i <language> ?l .  
  FILTER ?l != 'English'  
}
```

(Clustered) property table

```
SELECT T.subject, T.object  
FROM TRIPLES AS T,  
      PROPTABLE AS P  
WHERE T.subject == P.subject  
      AND P.copyright < 1990  
      AND T.property = 'language'  
      AND T.object != 'English'
```

Property-class table

```
(SELECT T.subject, T.object  
 FROM TRIPLES AS T, BOOKS AS B  
 WHERE T.subject == B.subject  
       AND B.copyright < 1990  
       AND T.property = 'language'  
       AND T.object != 'English')  
UNION  
(SELECT T.subject, T.object  
 FROM TRIPLES AS T, CDS AS C  
 WHERE T.subject == C.subject  
       AND C.copyright < 1990  
       AND T.property = 'language'  
       AND T.object != 'English')
```

Query execution complicates very quickly

Vertical Partitioning

[Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007]

Decomposed storage model

- a.k.a. column store

Type	
ID1	BookType
ID2	CDType
ID3	BookType
ID4	DVDType
ID5	CDType
ID6	BookType

Author	
ID1	"Fox, Joe"

Title	
ID1	"XYZ"
ID2	"ABC"
ID3	"MNO"
ID4	"DEF"
ID5	"GHI"

Artist	
ID2	"Orr, Tim"

Copyright	
ID1	"2001"
ID2	"1985"
ID5	"1995"
ID6	"2004"

Language	
ID2	"French"
ID3	"English"

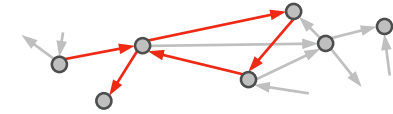
- One table per property
- Two columns pre table: subject and object
- Sorted by subject
 - Quick look-up of a subject
 - Fast sort-merge joins to reconstruct entities
- Multivalued attributes possible
- Column-oriented compression techniques can be applied
- Queries that have unspecified property are problematic

Basic Graph Concepts

The following assumes a directed graph $G(V, E \subset V \times V)$.
Concepts can be transferred to other kind of graphs.

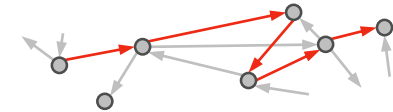
Path

- Sequence of edges connecting vertices
 - $w_{v_1, v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$ with $e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n$
 - n is the length of the path



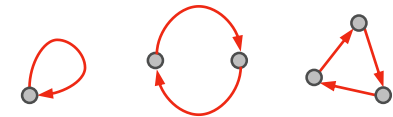
Simple Path

- Path connecting distinct vertices, i.e. path without cycles
 - $p_{v_1, v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$
with $\forall v_i, v_j: i \neq j \rightarrow v_i \neq v_j$ and $e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n$



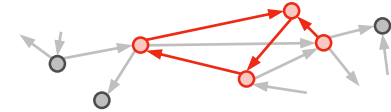
Cycle

- Walk with $v_1 = v_n$ is a cycle
- Graph is acyclic iff $\nexists v \in V: w_{v,v} \in G$



Subgraph

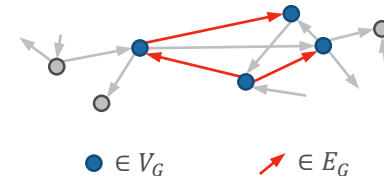
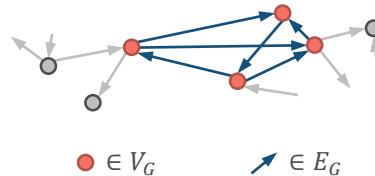
- Graph whose vertices and edges are fully part of a larger graph is a subgraph of that larger graph
- $G(V_G, E_G)$ is a subgraph of $H(V_H, E_H)$ if $V_G \subseteq V_H$ and $E_G \subseteq E_H$
- G is a subgraph of H denoted as $G \subseteq H$



Vertex- / edge-induced subgraph

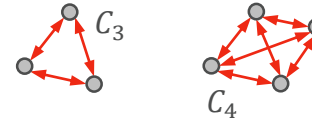
- Subgraph defined by a subset of vertices / edges of a given graphs
- A subgraph of $H(V_H, E_H)$ induced by vertex set
is a graph $G(V_G, E_G)$ with

$$V_G \subseteq V_H \quad / \quad E_G \subseteq E_H$$
$$E_G = \{(x, y) \in E_H \mid x, y \in V_G\} \quad / \quad V_G = \{v \in V_H \mid (v, \cdot) \in E_G \wedge (\cdot, v) \in E_G\}$$



Cliques

- Fully edge-connected set of vertices
- $C \subseteq V$ with $\forall v_i, v_j \in C: (v_i, v_j) \in E$



Connected components

- Fully path-connected set of vertices, i.e. there exists a path between every pair of vertices
- $C \subseteq V$ with $\forall v_i, v_j \in C \exists p \in G: p = v_i, \dots, v_j$
- With considering direction: strongly connected
- Without considering direction: weakly connected
 - Directed edge are treated like undirected once



Degree (or valency)

- In/out degree of a vertex: Number of incoming/outgoing edges of that vertex

- $\deg_{out}(v) = |\{(v, u) \in E\}|$

- $\deg_{in}(v) = |\{(u, v) \in E\}|$

- $\deg(v) = \deg_{out}(v) + \deg_{in}(v)$

- Degree of a graph: Maximum vertex degree

- $\deg(G) = \Delta(G) = \max_{v \in V} \deg(v)$



Degree distribution

- probability distribution of vertex degree in a graph

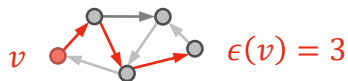
Distance

- Distance between two vertices in a graph is number of edges in a shortest path connecting them

- $d(v, u) = \min_{p_{v,u} \in G} |p_{v,u}|$

Eccentricity

- Greatest distance of vertex to any other vertex
- Longest shortest path starting from v
- $\epsilon(v) = \max_{u \in V} d(v, u)$



Radius

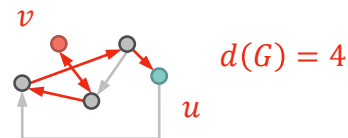
- Minimum eccentricity of any vertex in the graph
- $r(G) = \min_{v \in V} \epsilon(v) = \min_{v \in V} (\max_{u \in V} d(v, u))$

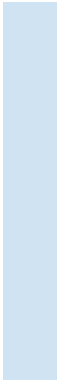
Average distance (average shortest path)

- Average eccentricity of any vertex in the graph
- $L(G) = \frac{\sum_{v \in V} \epsilon(v)}{|V|}$

Diameter

- Maximum eccentricity of any vertex in the graph
- $d(G) = \max_{v \in V} \epsilon(v) = \max_{v \in V} (\max_{u \in V} d(v, u))$

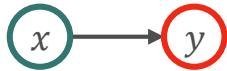




Neo4j Cypher

Graph Pattern Matching

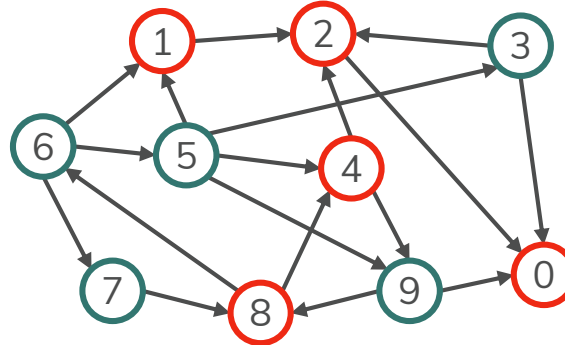
Subgraph pattern p



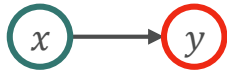
Graph with place holders x and y

Matching p on G

Finds all subgraphs in G that fit to p



Subgraph pattern p



Graph with place holders x and y

Intuitively, how many matches does p have on G ?

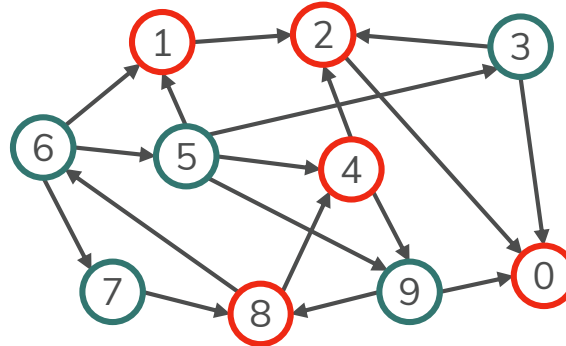
Number of Results

A	5
B	8
C	10
D	14



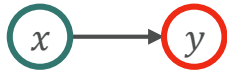
Matching p on G

Finds all subgraphs in G that fit to p



Graph Pattern Matching

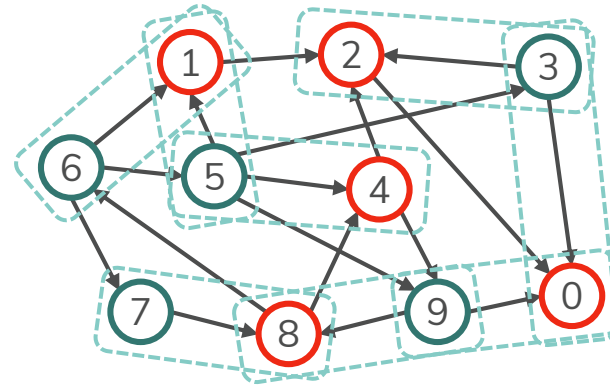
Subgraph pattern p



Graph with place holders x and y

Matching p on G

Finds all subgraphs in G that fit to p



[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>]

Vertex pattern

- `()` unidentified vertex
- `(matrix)` vertex identified by variable *matrix*
- `(:Movie)` unidentified vertex with label *Movie*
- `(matrix:Movie:Action)` vertex with labels *Movie* and *Action* identified by variable *matrix*
- `(matrix:Movie {title: "The Matrix"})` + property *title* equal the string "The Matrix"
- `(matrix:Movie {title: "The Matrix", released: 1997})`
+ property *released* equal the integer 1997

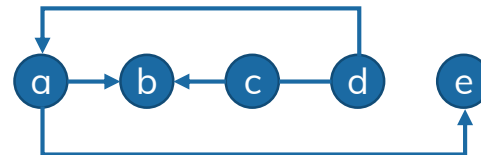
Edge pattern

- `--` unidentified undirected edge (matches either direction)
- `-->` unidentified directed edge
- `-[role]->` directed edge identified by variable *role*
- `-[:ACTED_IN]->` unidentified directed edge with label *ACTED_IN*
- `-[role:ACTED_IN]->` directed edge with label *ACTED_IN* identified by variable *role*
- `-[role:ACTED_IN {roles: ["Neo"]}]->` + property *roles* contains the string "Neo"

[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>]

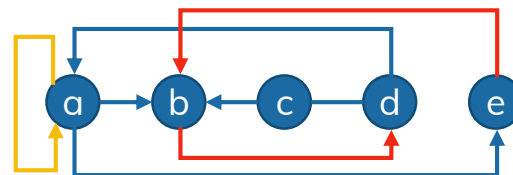
Path patterns

- String of alternating vertex pattern and edge pattern
- Starting and ending with a vertex pattern
- $(a) \rightarrow (b) \leftarrow (c) \rightarrow (d) \rightarrow (a) \rightarrow (e)$
- `(keanu:Person:Actor {name: "Keanu Reeves"})
-[role:ACTED_IN {roles: ["Neo"]}]-> (matrix:Movie {title: "The Matrix"})`



Graph patterns

- One or multiple path patterns
- Path patterns should have at least one shared variable
- Without shared variable graph pattern is disconnected
 - Results in a cross-product of the results for connected sub patterns
 - Quadrating blow up in result size and computational complexity
- $(a) \rightarrow (b) \leftarrow (c) \rightarrow (d) \rightarrow (a) \rightarrow (e), (e) \rightarrow (b) \rightarrow (d), (a) \rightarrow (a)$



[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>]

Which are valid pattern?

- A: (a,b:Movie)-[:SHOWN_IN]->(e),(f)
- B: (a:Movie)-[:SHOWN_IN]->()
- C: (:Movie)-[:SHOWN_IN]->
- D: ()<--(a:Movie)



Pattern Syntax

[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>]

Which are valid pattern?

- A: (a,b:Movie)-[:SHOWN_IN]->(e),(f)
- B: (a:Movie)-[:SHOWN_IN]->()
- C: (:Movie)-[:SHOWN_IN]->
- D: ()<--(a:Movie)

[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>]

Which are valid pattern?

- A: `(a,b:Movie)-[:SHOWN_IN]->(e),(f)`
- B: `(a:Movie)-[:SHOWN_IN]->()`
- C: `(:Movie)-[:SHOWN_IN]->`
- D: `()<--(a:Movie)`

Which pattern specifies a loop?

- A: `(a:Movie {name: "Matrix"})-->(a)`
- B: `(a:Movie {name: "Matrix"})-->(b:Movie {name: "Matrix"})`
- C: `(a:Movie {name: "Matrix"})-->(a:Movie {name: "Matrix"})`
- D: `(a:Movie {name: "Matrix"})-->({name: "Matrix"})`



[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>]

Which are valid pattern?

- A: `(a,b:Movie)-[:SHOWN_IN]->(e),(f)`
- B: `(a:Movie)-[:SHOWN_IN]->()`
- C: `(:Movie)-[:SHOWN_IN]->`
- D: `()<--(a:Movie)`

Which pattern specifies a loop?

- A: `(a:Movie {name: "Matrix"})-->(a)`
- B: `(a:Movie {name: "Matrix"})-->(b:Movie {name: "Matrix"})`
- C: `(a:Movie {name: "Matrix"})-->(a:Movie {name: "Matrix"})`
- D: `(a:Movie {name: "Matrix"})-->({name: "Matrix"})`

Matching, Filtering, Result Definition

Matching

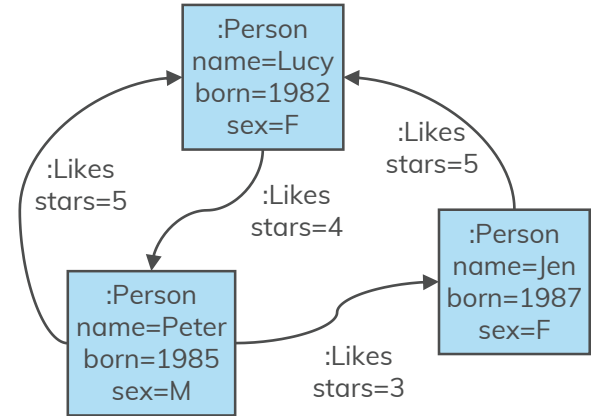
[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/match/>]
[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]

MATCH clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH clauses

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

Example



Number of Results	
A	2
B	3
C	4
D	5



Matching

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/match/>]
[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]

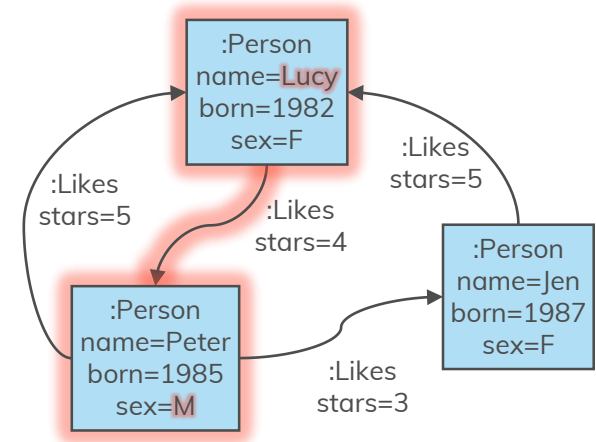
MATCH clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH clauses

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

Example



Matching

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/match/>]
[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]

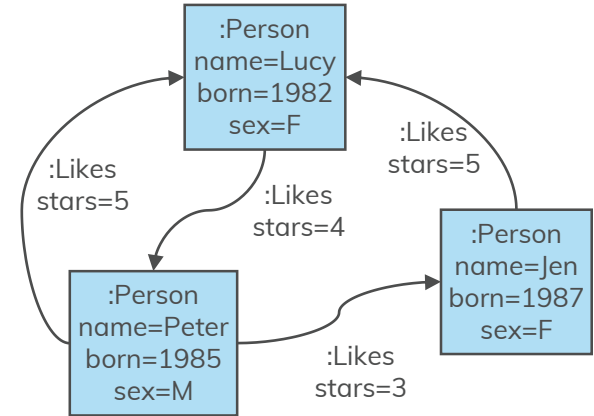
MATCH clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH clauses

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

```
MATCH (p:Person)-[:Likes]->(:Person)-[:Likes]->(fof:Person)
RETURN p.name, fof.name
```

Example



p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

Number of Results	
A	2
B	3
C	4
D	5



Matching

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/match/>]
[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]

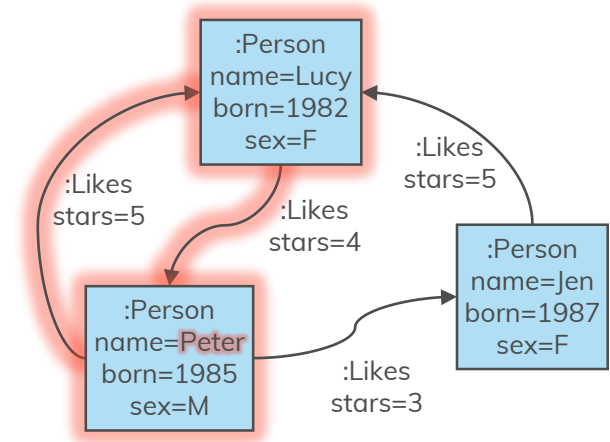
MATCH clause

- Primary way of getting data from a Neo4j database
- Allows you to specify a subgraph pattern
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH clauses

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

```
MATCH (p:Person)-[:Likes]->(:Person)-[:Likes]->(fof:Person)
RETURN p.name, fof.name
```

Example



p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

p.name	fof.name
Lucy	Jen
Peter	Lucy
Peter	Peter
Jen	Peter
Lucy	Lucy

Optional Match

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/optional-match/>]

OPTIONAL MATCH clause

- Matches patterns against your graph database, just like MATCH
- Matches the complete pattern or not
- If no matches are found, OPTIONAL MATCH will use NULLs as bindings
- Like relational outer join

Example:

- MATCH (a:Movie)
 OPTIONAL MATCH (a)-[:WROTE]-(x)
 RETURN a.title, x.name



\$ MATCH (a:Movie) OPTIO...

Rows

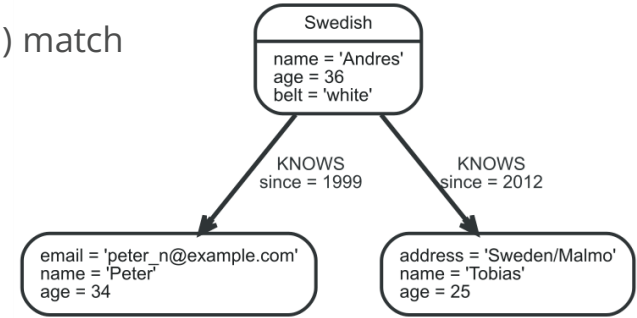
Code

a.title	x.name
The Matrix	null
The Matrix Reloaded	null
The Matrix Revolutions	null
The Devil's Advocate	null
A Few Good Men	Aaron Sorkin
Top Gun	Jim Cash
Jerry Maguire	Cameron Crowe
Stand By Me	null
As Good as It Gets	null
What Dreams May Come	null
Snow Falling on Cedars	null
You've Got Mail	null
Sleepless in Seattle	null
Joe Versus the Volcano	null
When Harry Met Sally	Nora Ephron
That Thing You Do	null
Returned 41 rows in 40 ms.	

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/>]

WHERE clause

- After an (OPTIONAL) MATCH, it adds constraints to the (optional) match
 - WHERE becomes part of the pattern
 - After a WITH, it just filters the result
 - Syntax: WHERE <expression>
-
- MATCH (n)
WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = "Tobias")
OR NOT (n.name = "Tobias" OR n.name="Peter")
RETURN n



n

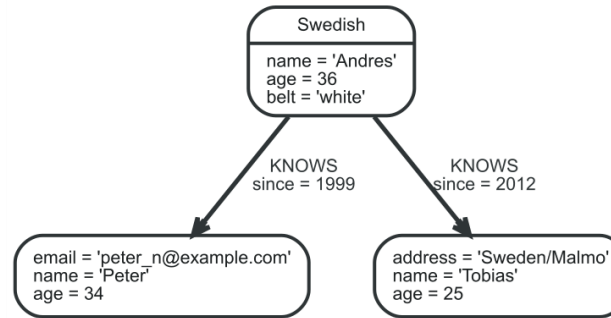
Node[0]{name:"Andres",age:36,belt:"white"}

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

Filtering

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/>]



- Filter on node label

`MATCH (n) WHERE n:Swedish RETURN n`

n

Node[0]{name:"Andres",age:36,belt:"white"}

- ... on node property

`MATCH (n) WHERE n.age < 30 RETURN n`

n

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

- ... on relationship type

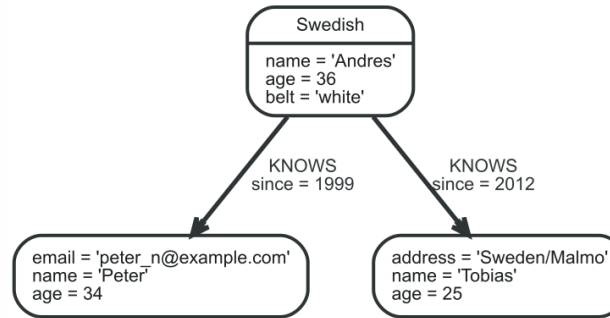
`MATCH (n)-[k]->(f) WHERE type(k) = 'KNOWS'
AND k.since < 2000 RETURN f`

n

Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

Filtering

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/>]



- ... on lists

MATCH (n) WHERE a.name IN ["Peter", "Tobias"] RETURN n

n

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

- ... on string properties
 - prefixes
 - suffixes
 - infixes
 - regular expressions

MATCH (n) WHERE n.name = 'Peter' RETURN n

MATCH (n) WHERE n.name STARTS WITH 'Pet' RETURN n

MATCH (n) WHERE n.name ENDS WITH 'ter' RETURN n

MATCH (n) WHERE n.name CONTAINS 'ete' RETURN n

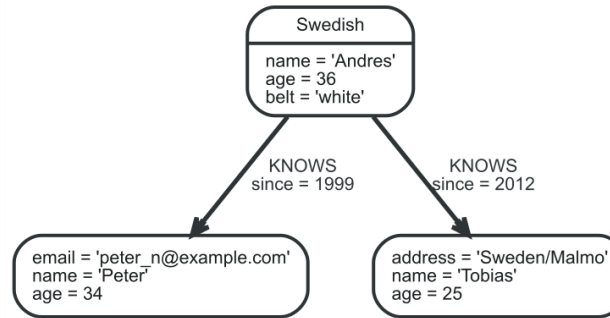
MATCH (n) WHERE n.name =~ 'P[et]+r?' RETURN n

n

Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

Filtering

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/>]



- ... on property existence
false is default for missing values

```
MATCH (n) WHERE exists(n.belt) RETURN n  
MATCH (n) WHERE NOT n.belt = 'white' RETURN n
```

n

Node[0]{name:"Andres",age:36,belt:"white"}

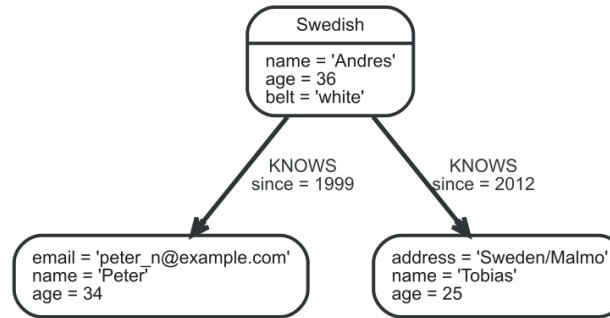
- ... on property non-existence MATCH (n) WHERE NOT exists(n.belt) RETURN n
MATCH (n) WHERE n.belt IS NULL RETURN n

n

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

Filtering

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/>]



- Using patterns for filtering
`MATCH (tobias { name: 'Tobias' }),(others)`
`WHERE others.age > 30 AND (tobias)<--(others)`
`RETURN others`
- ... with negation
`MATCH (persons),(peter { name: 'Peter' })`
`WHERE NOT (persons)-->(peter)`
`RETURN persons`
- ... on existence
`MATCH (person)`
`WHERE EXIST((person)-->())`
`RETURN person`

n

Node[0]{name:"Andres",age:36,belt:"white"}

n

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

other

Node[0]{name:"Andres",age:36,belt:"white"}

Projection

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]
[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/expressions/>]

RETURN clause

- Defines what to include in the query result set
- Comparable with relational projection
- Only once per query
- Allows to return nodes, edges, properties, or any expressions
- Column can be rename using AS <new name>

Example

- MATCH (n)
RETURN n, "node " + id(n) + " is " +
CASE WHEN n.title IS NOT NULL THEN "a Movie"
WHEN EXISTS(n.name) THEN "a Person"
ELSE "something unknown"
END AS about



\$ MATCH (n) RETURN n, "node "...					
 Graph	n	about			
 Rows	released1999 titleThe Matrix taglineWelcome to the Real World	node 175 is a Movie			
	born1964 nameKeanu Reeves	node 176 is a Person			
	born1967 nameCarrie-Anne Moss	node 177 is a Person			
	born1961 nameLaurence Fishburne	node 178 is a Person			
Returned 174 rows in 46 ms.					

Projection

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]

RETURN clause

- Return nodes

```
MATCH (n { name: "B" }) RETURN n
```

n
Node[1]{name:"B"}

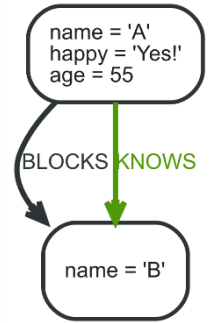
- Return relationships MATCH (n { name: "A" })-[r:KNOWS]->(c) RETURN r

r
:KNOWS[0]{}

- Return property

```
MATCH (n { name: "A" }) RETURN n.name
```

n.name
"A"



Projection

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]

RETURN clause

- Column alias

```
MATCH (a { name: "A" })  
RETURN a.age AS SomethingTotallyDifferent
```

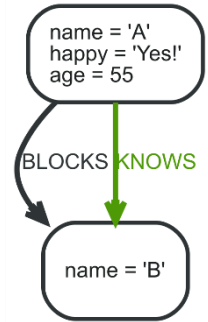
SomethingTotallyDifferent

55

- Return all elements

```
MATCH p=(a { name: "A" })-[r]->(b) RETURN *
```

a	b	p	r
Node[0]{name:"A",happy:"Yes!",age:55}	Node[1]{name:"B"}	[Node[0]{name:"A",happy:"Yes!",age:55},:BLOCKS[1]{},Node[1]{name:"B"}]	:BLOCKS[1]{}
Node[0]{name:"A",happy:"Yes!",age:55}	Node[1]{name:"B"}	[Node[0]{name:"A",happy:"Yes!",age:55},:KNOWS[0]{},Node[1]{name:"B"}]	:KNOWS[0]{}



Projection

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>]
[<http://neo4j.com/docs/developer-manual/current/cypher/syntax/expressions/>]

RETURN clause

- Optional properties

MATCH (n) RETURN n.age

n.age
55
<null>

- Other expressions

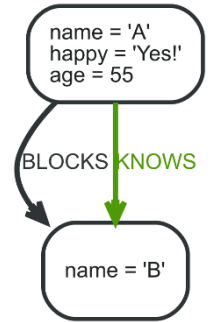
MATCH (a { name: "A" })
RETURN a.age > 30, "I'm a literal", (a)-->()

a.age > 30	"I'm a literal"	(a)-->()
true	"I'm a literal"	[[Node[0]{name:"A",happy:"Yes!",age:55},:BLOCKS[1]{}],Node[1]{name:"B"}],[Node[0]{name:"A",happy:"Yes!",age:55},:KNOWS[0]{}],Node[1]{name:"B"}]]

- Unique results

MATCH (a { name: "A" })-->(b) RETURN DISTINCT b

b
Node[1]{name:"B"}



Result Modification

[<http://neo4j.com/docs/developer-manual/current/cypher/clauses/order-by/>]

ORDER BY clause

- Sub-clause following RETURN or WITH
- Specifies that the output should be sorted and how
- Order by property
MATCH (n) RETURN n **ORDER BY n.name**
- Order by multiple properties
MATCH (n) RETURN n **ORDER BY n.age, n.name**
- Order nodes in descending order
MATCH (n) RETURN n **ORDER BY n.name DESC**
- Cannot sort on nodes or relationships, just on properties on these
- NULL will come at the end in ascending order (ASC), and first in descending order (DESC)



n
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}
Node[2]{name:"C",age:32,length:185}



n
Node[2]{name:"C",age:32,length:185}
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}

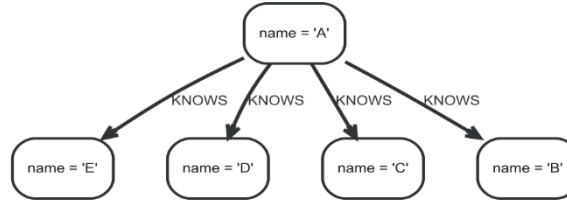


n
Node[2]{name:"C",age:32,length:185}
Node[1]{name:"B",age:34}
Node[0]{name:"A",age:34,length:170}

Result Modification

[\[http://neo4j.com/docs/developer-manual/current/cypher/clauses/limit/\]](http://neo4j.com/docs/developer-manual/current/cypher/clauses/limit/)
[\[http://neo4j.com/docs/developer-manual/current/cypher/clauses/skip/\]](http://neo4j.com/docs/developer-manual/current/cypher/clauses/skip/)

LIMIT clause



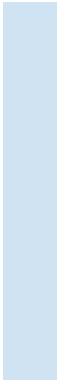
- Constrains the number of rows in the output
- Accepts any expression that evaluates to a positive integer
- Expression cannot refer to nodes or relationships
- Return first from the top
`MATCH (n) RETURN n ORDER BY n.name LIMIT 3`
- Return first from expression
`MATCH (n) RETURN n ORDER BY n.name LIMIT toInt(3 * rand()) + 1`

n
Node[0]{name:"A"}
Node[1]{name:"B"}
Node[2]{name:"C"}

SKIP clause

- Defines from which row to start including the rows in the output
- Result set will get trimmed from the top
- Same rules as for LIMIT
- Skip first three
`MATCH (n) RETURN n ORDER BY n.name SKIP 3`

n
Node[3]{name:"D"}
Node[4]{name:"E"}



Common Use Cases - Graph Clustering

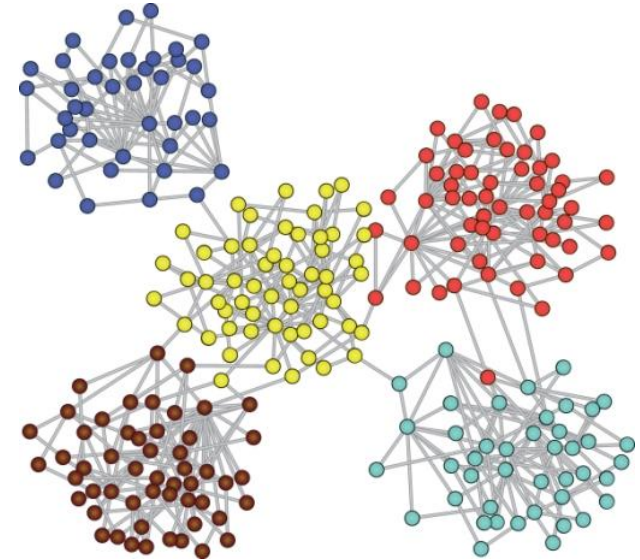
Division of a graph into a set of subgraphs

- such that there are a dense set of edges within every cluster and few edges between clusters

Min-max cut methods

- Idea
 - Seeks to partition a graph into clusters A and B
 - Minimizing the number of connections between A and B
 - Maximizing the number of connections within each
- Used measure
 - *cut* is the number of edges that would have to be removed to isolate the vertices in A from those in B
 - Algorithms search clusters that minimize *cut*
- Needs addition criteria: balanced cluster size
- Run procedure recursively to create $k > 2$ clusters

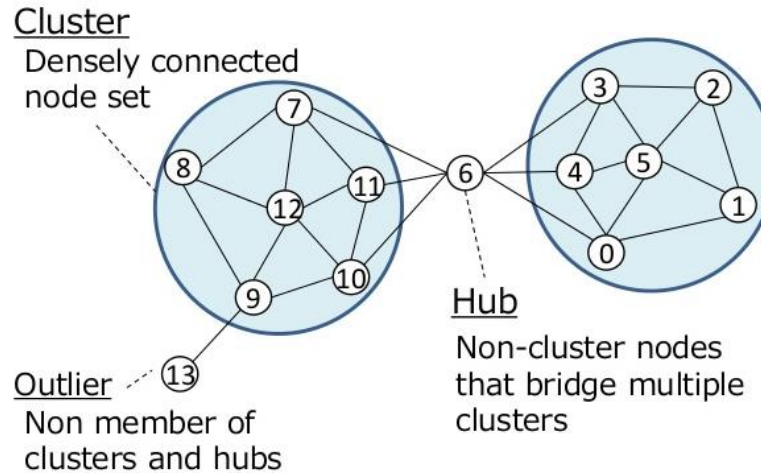
Other quality measure: Modularity



[<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2542420/>]

Structure-connected clusters

- Extends notion of a density-based cluster to graph
- Can distinguish good clusters, hubs, and outliers



Algorithms

[<https://www.slideshare.net/LazyShion/scan-efficient-algorithm-for-finding-clusters-hubs-and-outliers-on-largescale-graphs-vldb-2015>]

- SCAN (original), LinkSCAN* (uses edge sample), SCAN++ (more efficient)

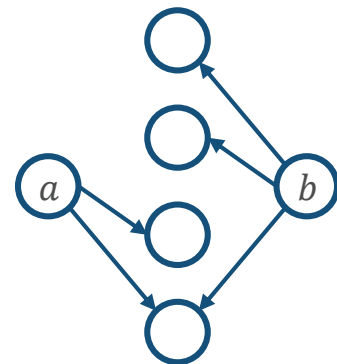
[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Intuition

- Two person are similar if they know the same people
- A group of similar people forms a community (a cluster)

Distance and similarity

- Neighborhood of a vertex: $\Gamma(v) = \{w \in V \mid (v, w) \in E\} \cup \{v\}$
- Distance $\sigma(v, w)$ of two adjacent vertices v and w is
 - Number of common neighbors
 - Normalized by the geometric mean of the two neighborhoods' size
 - $$\sigma(v, w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)| \cdot |\Gamma(w)|}}$$
 - Distance of non-adjacent vertices is infinite, i.e. not considered
- Two vertices with similarity above a given threshold ϵ considered structurally similar
 - ϵ -neighborhood of v are all structurally similar neighbors of v
 - $N_\epsilon(v) = \{w \in \Gamma(v) \mid \sigma(v, w) \geq \epsilon\}$
 - ϵ defines a volume



$$\sigma(a, b) = \frac{1}{\sqrt{2 \cdot 3}} = 0,408 \dots$$

[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Cluster cores

- Cores are the vertices from where cluster “grow”
- Vertex with more than μ structural similar neighbors are core vertices
 - $CORE_{\epsilon,\mu}(v) \Leftrightarrow |N_{\epsilon}(v)| \geq \mu$
 - μ is a threshold to the number of elements per ϵ -volume
 - μ and ϵ together define a density threshold

Reachability

- All vertices in the ϵ -neighborhood (structural similar neighbors) of a core are directly reachable
 - Direct structure reachability
 - $DirREACH_{\epsilon,\mu}(v, w) \Leftrightarrow CORE_{\epsilon,\mu}(v) \wedge w \in N_{\epsilon}(v)$
- Chains of directly reachable vertices are reachable
 - Structure reachability is transitive closure of direct structure reachability
 - $REACH_{\epsilon,\mu}(v, w) \Leftrightarrow \exists u_1, \dots, v_n \in V: u_1 = v \wedge u_n = w \wedge \forall i \in [1, n - 1]: DirREACH_{\epsilon,\mu}(v_i, v_{i+1})$

[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Connectivity

- Two vertices that are reachability from the same core are connected
 - Structure connectivity
 - $CONNECT_{\epsilon,\mu}(v, w) \Leftrightarrow \exists u: REACH_{\epsilon,\mu}(u, v) \wedge REACH_{\epsilon,\mu}(u, w)$

Cluster

- All vertices that are connected to one another belong to the same cluster
 - $CLUSTER_{\epsilon,\mu}(C) \Leftrightarrow$
 - (1) Connectivity: $\forall v, w \in C: CONNECT_{\epsilon,\mu}(v, w)$
 - (2) Maximality: $\forall v, w \in V: v \in C \wedge REACH_{\epsilon,\mu}(v, w) \Rightarrow w \in C$

Clustering

- Set of all cluster of given graph forms a clustering
 - $CLUSTERING_{\epsilon,\mu}(P) \Leftrightarrow P = \{C \subseteq V \mid CLUSTER_{\epsilon,\mu}(C)\}$

Hub

- A hub is a vertex not belonging to a cluster but has neighbors in two or more clusters of a clustering
 - $HUB_{\epsilon, \mu}(v) \Leftrightarrow$
 - (1) v is not a member of any cluster: $\forall C \in P: v \notin C$
 - (2) v bridges different clusters: $\exists X, Y \in P: X \neq Y \wedge X \cap Y \cap \Gamma(v) \neq \emptyset$

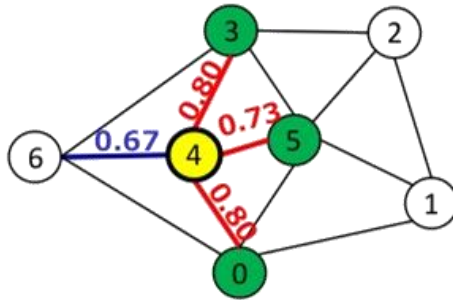
Outlier

- A outlier is a vertex not belonging to a cluster and it all it neighbors either belong to only one cluster or do not belong to any cluster
 - $OUTLIER_{\epsilon, \mu}(v) \Leftrightarrow$
 - (1) v is not a member of any cluster: $\forall C \in P: v \notin C$
 - (2) v does not bridge different clusters: $\forall X, Y \in P: X \neq Y \wedge X \cap Y \cap \Gamma(v) = \emptyset$

[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Structure-connected clusters

- Example: $\epsilon = 0.7$, $\mu = 2$

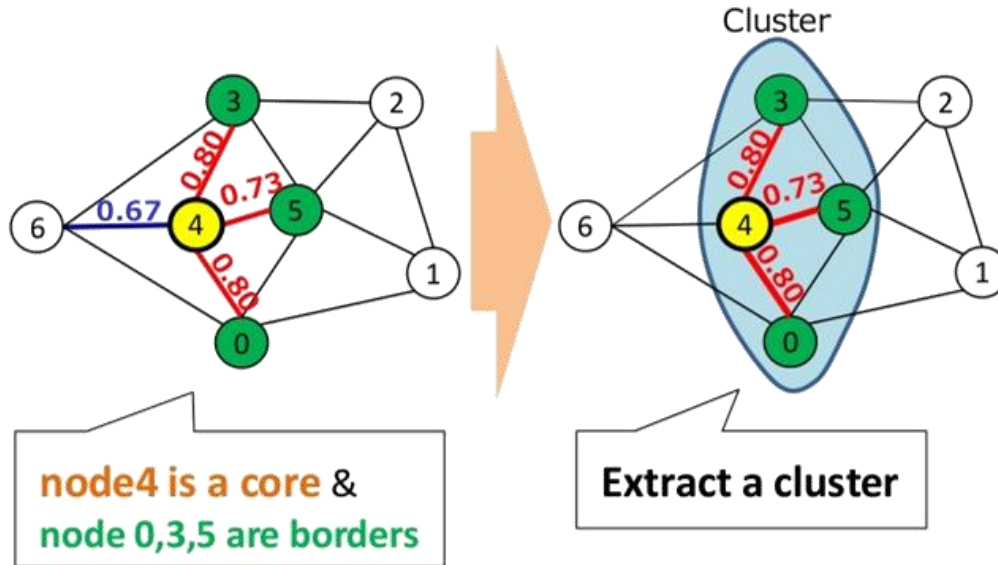


node4 is a core &
node 0,3,5 are borders

[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Structure-connected clusters

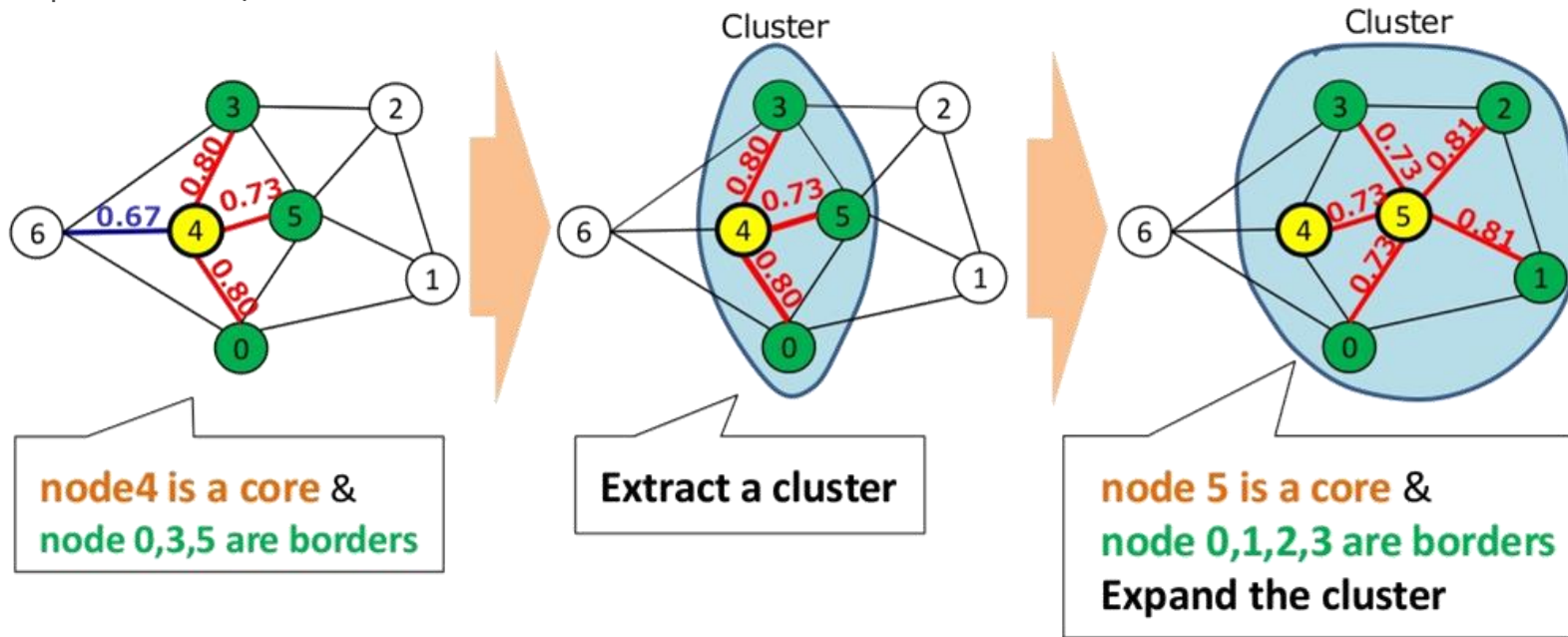
- Example: $\epsilon = 0.7$, $\mu = 2$



[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Structure-connected clusters

- Example: $\epsilon = 0.7$, $\mu = 2$



[Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007]

Algorithm

```

scan( $G(V, E), \epsilon, \mu$ )
    // initially all vertices in  $V$  have  $v.\text{clusterId}$  is undefined;
    for each  $v \in V$  |  $v.\text{clusterId}$  is undefined do
        // STEP 1. check whether  $v$  is a core;
        if  $\text{CORE}_{\epsilon, \mu}(v)$  then
            // STEP 2.1. if  $v$  is a core, a new cluster is expanded;
             $C := \text{new clusterId}; // \neq OUT$ 
            insert all  $x \in N_{\epsilon}(v)$  into queue  $Q$ ;
            while  $Q \neq \emptyset$  do
                 $y := Q.\text{first}$ 
                for each  $x \in V$  |  $\text{DirREACH}_{\epsilon, \mu}(y, x)$  do
                    if ( $x.\text{clusterId}$  is undefined) then
                        insert all  $x$  into queue  $Q$ 
                    if ( $x.\text{clusterId}$  is undefined OR  $x.\text{clusterId} = OUT$ ) then
                         $x.\text{clusterId} := C$ 
            else
                // STEP 2.2. if  $v$  is not a core, it is labeled as outlier
                 $x.\text{clusterId} := OUT$ 
        // STEP 3. further classify outliers
    for each  $v \in V$  |  $v.\text{clusterId} = OUT$  do
        if  $\exists x, y \in \Gamma(v): x.\text{clusterId} \neq y.\text{clusterId}$  then
             $x.\text{clusterId} := HUB$ 
    
```